NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

OBJECT-ORIENTED PLAN REPRESENTATION FOR THE OMWG C2 OBJECT SCHEMA

by

Robert M. Reeves

March, 1997

Principal Advisor: Dan Boger

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE OBJECT-ORIENTED PLAN REPRESENTATION FOR THE OMWG C2 OBJECT SCHEMA	5. FUNDING NUMBERS
6. AUTHOR: LT Robert M. Reeves, USN	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The purpose of this thesis is to examine current Command and Control planning methods and to aid in the furtherance of the Object Model Working Group's (OMWG) Core Plan Representation. Chapter I introduces the discipline of planning and its history. Chapter II discusses the theory and practice of modern Object-Oriented modeling. The structure and conventions of object programming are covered as well as a method for information system abstraction. Chapter III covers the background of current Command and Control systems and gives a report on the OMWG efforts in creation of an Object Schema for Command and control. Chapter IV presents the author's submission for an Object-Oriented representation of the COMSUBPAC OPLAN 5050 based on the Core Plan Representation (CPR).

14. SUBJECT TERMS: Object-Oriented, Plan, Schema, C2, C4I.			15. NUMBER OF PAGES 68
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

OBJECT-ORIENTED PLAN REPRESENTATION FOR THE OMWG C2 OBJECT SCHEMA

Robert M. Reeves
Lieutenant, United States Navy
B.S., United States Naval Academy, 1989

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March, 1997

Author: Robert Reeves
LT Robert M. Reeves

Approved by: Dan C Boger
Dan Boger, Principal Advisor

Tung Bui, Associate Advisor

for
Reuben Harris, Chairman
Department of Systems Management

ABSTRACT

The purpose of this thesis is to examine current Command and Control planning methods and to aid in the furtherance of the Object Model Working Group's (OMWG) Core Plan Representation. Chapter I introduces the discipline of planning and its history. Chapter II discusses the theory and practice of modern Object-Oriented modeling. The structure and conventions of object programming are covered as well as a method for information system abstraction. Chapter III covers the background of current Command and Control systems and gives a report on the OMWG efforts in creation of an Object Schema for Command and control. Chapter IV presents the author's submission for an Object-Oriented representation of the COMSUBPAC OPLAN 5050 based on the Core Plan Representation (CPR).

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Planning for Command and Control in the United States Military has evolved significantly over the last 200 years. The recognition of a need for formalized contingency planning, as well as for development of courses of action during a conflict, was born out of the realization that relying solely on an individual unit's sense of "Commanders Intent" was gravely insufficient. Command and control in theory has always been a primary element of military operations. It serves the purpose, during all facets of military operations from peace to war, of controlling not only the actions and tactics of our own forces, but affecting and limiting the options of our enemies. (Naval Doctrine Publication 6, 1995, pg. 3)

Command and Control allows the military commander to understand the situation, select a course of action, issue orders, and monitor the outcomes in order to better select subsequent courses of action. This can be represented by what is referred to as the OODA loop. In this loop, the commander first *observes* the situation at hand. He or she needs a great deal of information at this point, and at many levels of granularity, from specific resources to overall battle space conditions. After observing, the commander will then *orient* the forces involved. This requires synthesizing the information gathered during observation and, often times, making an educated guess as to how to position forces and resources to support possible future actions. Third, the commander will use his observations and orientation to make the best *decision* possible, after which the forces will *act* upon that decision. At this point, the process loops so that the results of the action can then be observed, beginning the OODA loop over again. (NDP 6, 1995, pg.18)

Command and Control is key to using existing mobility and firepower in the most efficient way to ensure success in operations. The complexity of the battle space due to emerging technology has increased dramatically in recent years. During the Battle of Trafalgar in 1805, Admiral Horatio Nelson used only three general tactical flag-hoist signals to maneuver the entire British Fleet. By comparison, during Operation Desert Storm, General Schwarzkopf's U.S. Central Command used up to 700,000 phone calls and 152,000 radio messages per day to coordinate the actions of U.S. and Coalition forces (NDP 6, 1995, p. 4).

As defined by Joint Publication 1-02, Command and Control is

> the exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission. Command and Control functions are performed through and arrangement of personnel, equipment, communications, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission.

A Command and Control System includes equipment, people, communications, and procedures needed by the commander in order to plan, direct and control operations and forces. Currently, commanders can exercise two methods of control - detailed control and mission control. Under detailed control, Command and Control is centralized, and orders and plans are explicit, such as during tactical maneuvering. Detail control, however, is not particularly suited for rapidly changing situations, due to the time-consuming vertical information flow it requires. Mission control attempts to mitigate the effects of uncertainty and time by creating a quick response system. Mission control in essence is more akin to the

old system of "commander's intent" in which individual commanders were well briefed in the desired end state of a situation, the acceptable methods for achieving that state, and then allowed to use their own judgement during actual tactical engagements. Mission Control is also known widely as Command by Negation. This means essentially that the lower level commanders are trusted to perform their leader's intent in the way they feel is most appropriate, and they will only be corrected if the higher level commander feels it is absolutely necessary. The emergence of modern C4I systems seems to blur the distinction between these two forms of control. The near instantaneous flow of information from the battlefield to high level commanders will tempt many to employ detail control. In a recent missile firing exercise, for instance, the President of the United States was actually on-line with the firing platform, and was directing the Squadron Commodore in the performance of his exercise. This sort of blatant micro-management is a very real and serious issue that must be discussed soon in order to maintain the integrity of modern war-fighting concepts. Senior Officers train for years to attain the knowledge and sense of battlespace necessary for optimal performance of warfighting tactics, and second-guessing by off-scene entities is destructive to efficiency and morale.

A. PLANNING

Planning in the military has become relatively straightforward in practice. All commands, from the large Unified Commands down to small unit commands, have various and often well-defined interests. For instance, a COMDESRON will undoubtedly be interested in such activities as standard ship and squadron operations, weapons use,

emergency procedures, and submarine hunting and killing operations.

1. Current Planning Methods

The standard method for formalizing plans and procedures in the military is the creation of Operational Plans, or OPLANS. The current format for creating an OPLAN is delineated in Joint Pub-2. OPLANS come in widely varying scope. During an anticipated national engagement, such as Operation Desert Storm or the operations in Somalia, the supported Unified Commander, the Joint Chiefs of Staff, as well as other supporting commands, will create a large scale OPLAN that describes every facet of the operation. It includes analysis of the current situation and a detailed step-by-step description of how our desired end-state will be achieved. Included in most OPLANs is Time Phased Force Deployment Data (TPFDD). The TPFDD describes specifically how all resources from personnel to logistical equipment, will be moved to the theater of operations. Many other OPLANs have been developed that are of smaller scope, and serve to formalize common operations performed by various units, such as an air attack, or other carrier operations. The Submarine Search and Rescue procedures which are modeled in Chapter IV is such an OPLAN. OPLANs are typically followed by Operations Orders (OPORDS) which contain more specific commands to carry out the plans devised in the OPLAN.

OPLANs are mostly of interest to higher level commanders, and TPFDDs are typically of interest to logistics planners and transportation commands. Unfortunately, since both usually consist of huge printouts, neither are suitable for easy and quick extraction of data relevant to the reader's interest. It is not uncommon for OPLANs to be several hundred

pages long. The method used for OPLAN dissemination is the Joint Operations Planning and Execution System (JOPES) which was formerly carried on the World Wide Military Command and Control System (WWMCCS) Top Secret Network. WWMCCS host sites, and hence JOPES, were located at every Major Command, as well as many smaller forward commands, allowing Unified and other commanders access to OPLANs once they had been uploaded to the JOPES. The WWMCCS, which was officially shut-down in September of 1996, is replaced by the Global Command and Control System (GCCS), which represents a dramatic step forward in utilizing modern technology and information concepts.

2. Shortcomings

Analysis of the limitations of current planning methods involves examination of both the information itself and the system within which the information exists. There are several metrics available for us to rate the old planning systems and determine the potential gains to be made by restructuring and automating the planning process through an Object-Oriented system.

To judge the quality of information received from a system, we can use the following information attributes:

- completeness of the information
- accuracy of the information
- age of the information
- consistency of info across nodes in the command structure

- correctness of understandings
- correctness of consequences predicted
- fidelity of the directives to the decisions made

It is clear that OPLANs and TPFDDs in their current form are complete and accurate, since they are both an exhaustive culmination of planning effort from several agencies at the time that they are actually written. After today, however, OPLANs will tend to do poorly under the metrics of 'age of information' and 'consistency of information across nodes in the command structure'. This is a predictable effect merely because of the change in the nature of conflict. Operations Other Than War (OOTW) and other peacetime operations are highly dynamic, and an entire plan which made perfect sense only days ago can become completely non-applicable. Written OPLAN's other shortcoming has to do with the very nature of freeform written documentation: extraction of relevant information can be very difficult. Further, discrete documentation introduces the problems of version control, dissemination, and accountability.

The development and introduction of an Object-Oriented system to aid in the planning process will go far in the mitigation of the problems of outdated and inconsistent information. An automated OPLAN development tool based on objects will gain all the benefits afforded by this increasingly popular methodology. The OPLAN would become modular, and as such each section could be created by different sources simultaneously with all submissions and updates automatically integrated into the central repository. Access to OPLANs and

TPFDDs will be vastly simplified, and will be accomplished using the Common Object Request Broker Architecture (CORBA) method for distributing objects on a network.

The measure of an information system's quality can likewise be estimated using some or all of the following metrics:

- Reliability
- Maintainability
- Availability
- Operational Availability

Reliability is the probability that a system can perform its intended function for a specified interval under stated conditions. Maintainability is the measure of the ability for a system to be retained in or restored to a specified condition when maintenance is performed by personnel having specified skill levels, using prescribed procedures and resources. Availability is the measure of the degree to which a system is in an operable and committable state at the start of a mission when the mission is called for at a random time. The WWMCCS was run on a network of venerable Honeywell H6000 mainframes whose processing power is comparable to moderate performing x486 PCS. The software written to support JOPES and several other WWMCCS subfunctions was written in the late '70s, and the contractors responsible for maintenance of the systems have had to retain token personnel who would normally have long since retired, merely in order to have someone who remembers how the system was constructed. Since specific WWMCCS sites were typically

responsible for different parts of Command and Control operations, any down-time on a host cut off all access to the information kept solely at that host. While the reliability of these systems is fairly high due to their simplicity, their maintainability is extremely poor, and hence their availability factor has declined steadily over the years. Virtually all new C4I systems, such as the Joint Maritime Command Information System (JMCIS) and the Global Command and Control System (GCCS), while not strictly Object-Oriented systems, are based on a client-server architecture using either Unix or Windows/NT platforms. In either case, specific sites will now consist of small networks of several hosts, supporting redundancy in both system operation and database storage. System availability will be extremely high since the functions of any server on the network can be assumed by a backup server on the same local network. Maintainability is similarly high since all hardware and software in use will be current and contemporary, with a large support base of military as well as commercial sources.

B. SPECIFIC C2 FUNCTIONALITIES AND THEIR INTENT

The best way to create a new robust system which does as it was intended is to ignore existing automation efforts except for the required functions they represent. That is, systems like the GCCS implementation of JOPES should be examined only to see *what* is being done, but not *how* it is being done. It is more desirable to design an Object-Oriented system from the ground up by examining such intent documents as Copernicus and Forward ...From the Sea. These documents describe the latest methods of Command and Control and planning, as well as the functionality desired by the planners, and they do so without specifying the

systems to accomplish these aims.

1. Copernicus

Copernicus was written at a time when all the military services were struggling with the pressure to develop systems making best use of rapidly expanding computer technology. Copernicus was the Navy's document to design a user-centered, C4I information management architecture. The document is remarkable in that it was able to successfully describe the desired framework for C4I systems without being overly specific. Its main objective was to create a true sensor-to-shooter environment. There are five essential elements to Copernicus:

- Use common applications to blend tactical, operational, and administrative data to the war fighter.
- Favor an "information pull" environment rather than "information push" which can prevent information overload.
- Use multi-media representations where appropriate.
- Use common building blocks to promote modular hardware and software design.
- Provide a Common Operating Environment (COE) on workstations to increase user proficiency. (Copernicus ...Forward, 1994, pp. 2-3)

Although the original Copernicus document is now nearly five years old, the concepts it promoted have been followed to a large degree as evidenced by the Joint Maritime Command Information System (JMCIS) and the Global Command and Control System (GCCS).

2. Forward ...From the Sea

Forward ...From the Sea is the primary paper endorsed by the Chief of Naval Operations (CNO) as the Navy's vision and, hence, the Navy's specification for C4I systems now and in the future. It encompasses many of the concepts of Copernicus, but does so in a more detailed manner. As such it is more suggestive of the specific systems that are currently desired for the execution of C4I. In contrast to its specificity for systems, it is in most other ways a policy paper and contains many buzzwords as well as descriptions of what capabilities are generally expected when the appropriate computer systems are fielded.

3. C4I

Command, Control, Communications, Computers, and Intelligence (C4I) is the generic term used to describe the entire spectrum of systems used by the warfighter, although it is traditionally indicative of computerized and automated systems. With computer processing power doubling every 18 months, the opportunities presented by capitalizing on this field can easily keep the United States ahead in the Information Warfare game. The key to efficient allocation of resources is to identify ahead of time potential stovepipe systems and vigorously promote common interservice systems and applications.

a. Current Systems in Use

The most popular Command and Control automated system in the U.S. military at this time is the Global Command and Control System (GCCS). It enjoys strong support from the top. General J. Shalikashvili, Chairman of the Joint Chiefs of Staff has said, "I will support only one [Joint] Command and Control System." (GCCS, From Concept to Reality, 1994). This attitude by the military's senior leadership is a welcome endorsement of the military's commitment sound systems development practices and will hopefully lead to the elimination of older existing stovepipe systems which severely hamper efforts at interservice communication and cooperation.

II. OBJECT-ORIENTED TECHNOLOGY

Object-Oriented design of software is a complex discipline and has a great number of benefits over conventional sequential, or procedural programming. It is important to make the distinction between programming with objects and Object-Oriented programming. Many people can now write MS Windows applications with the help of development environments such as Delphi. Even though the 'widgets' such as scroll bars and dialog boxes are library objects, the underlying code written for these programs is still typically procedural. True Object-Oriented programming begins at the modeling stage, in which real world and conceptual objects are 'abstracted' from the desired system. The basics behind Object-Oriented programming are well documented, and most of the following explanations come from the book Designing Object-Oriented Software by Wirfs-Brock, Wilkerson, and Wiener.

A. OBJECTS

Real world problems can be inherently complex, and Object-Oriented modeling attempts to abstract out knowledge from the specifications and encapsulate it within objects. In order to find the objects and their connections, Object-Oriented programming ascertains what operations need to be performed and what information results from those operations. It then apportions responsibility for those operations and that information to objects. Each object knows how to perform its own operations and remember its own information. In essence, objects know how to be themselves.

1. Encapsulation

While an object knows how to be itself, it does not know how to be anything else. Some of the data in a system resides within certain of its objects and other data resides within other objects. Likewise, certain objects perform a particular function while other objects perform some other function. The act of grouping into a single object both data and the operations that affect that data is known as encapsulation. Encapsulation manages the complexity inherent in real-world problems by apportioning that complexity to the individual objects. The information encapsulated within an object can and sometimes should be hidden from external view. As a result, the information within an object often looks different from the outside than it does within the object itself. This means essentially that objects will tell the outside world what they can do and what they know, without divulging their entire contents. In this way other objects can determine how they can interact with it. How objects perform their functions is part of their private side. This consists of the specific coding within the methods of an object. How it performs the operations or computes the information is not a concern of other parts of the system. This principle is known as information-hiding. Using it, objects are free to change their private sides without affecting the rest of the system. Another object requesting an operation or some information acts like a manager. It specifies the job or asks for the information and then leaves. It does not care how the job is done or how the information is calculated. Objects know only what operations they can request other objects to perform. This also aids in the analysis and modeling of the system in that one can for the time being, concentrate on the abstract design without worrying about programming

specifics. Another gain comes later in the life of the system. One can change the internal representation of an object or implement a better algorithm for a specific operation without changing the object's public interface. Other objects which rely on that object's output or function will not be affected by the change. Encapsulation and information-hiding work together to isolate one part of the system from other parts, allowing code to be modified and bugs to be fixed without introducing unintended side effects.

2. Messages

Since objects can only access each other through their public interfaces, one object must access another object by sending it a message. Such access, known as message-send, is the only way objects interact. A message consists of the name of an operation and any required arguments. When one object sends a message to another object, the sender is requesting that the receiver of the message perform the named operation and perhaps return some information. When the receiver gets the message, it performs the request in the manner in which it was coded to do so. The set of messages to which an object can respond is known as the behavior of the object. An object has a set of internal messages which it uses for its own ends, in addition to those messages that are part of its public interface.

3. Classes and Instances

Individual objects can have similarities. An obvious example is the need in many applications for many of one type of object such as multiple boxes in a drawing or colors in a photo. Some objects in an application will behave differently from each other, and others will behave in a similar manner. Objects which share the same behavior are said to belong to

the same class. A class is a generic specification for an arbitrary number of similar objects. A class can be considered a template for a specific kind of object, and can be used to generate as many of these objects as needed. Classes allow us to abstract out the common behaviors of related objects, such as different types of chairs, and then to create objects that behave in that manner when we need them.

Objects that behave in a manner specified by a class are called instances of that class. All objects are instances of some class. Once an instance of a class is created, it behaves like all other instances of its class. A subclass is a class that inherits behavior from another class. A subclass usually adds its own behavior to define its own unique kind of object. For instance, all road vehicles have a set of common attributes, such as four wheels, an engine, a steering wheel, etc. However, if one wants to define a four-wheel drive vehicle, one can take the base class of vehicle to obtain our common attributes, and then add such attributes as a second differential, modified transmission, mud-shields and so on. Four-wheel drive vehicles are thus a subclass of vehicle.

a. Polymorphism

Limiting object access to message-send allows an abstraction method known as polymorphism. Polymorphism is the ability of two or more classes of object to respond to the same message in different ways. It allows us to recognize and exploit similarities between different classes of objects. Suppose a system allows the printing of several different types of file formats, and on some different type of printers. It would be unreasonable to expect the printing subsystem to learn different types of messages to cover every combination

of file type and printer type possible. By exploiting the polymorphism inherent in each file's behavior, it will print itself as needed, and on an appropriate printer with the needed capabilities. All the printer subsystem needs to do is monitor print queues and ask a file to print itself.

B. METHODS

When an object receives a message, it performs the requested operation by executing a method. A method is the specific algorithm executed in response to receiving a message whose name matches that of the method. A method is always part of the private side of an object in following with the concept of information-hiding. A method will be written in whatever computer language is being used on the system. While C++ is one of the most popular languages, it is still not a natural Object-Oriented programming language, like Smalltalk or Eiffel. As explained in the polymorphism section, many objects may have the same message name as part of their behavior, but employ completely different methods to respond to a request.

C. INHERITANCE

Object-Oriented programming languages support another abstraction mechanism: inheritance. Inheritance is the ability of one class to define the behavior and data structure of its instances as a superset of the definition of another class or classes. That is, we can define new classes that are similar to other classes except that they add some new specific behavior and/or attributes. This is the mechanism that allows the subclass example of the vehicle and four-wheel drive of above. Inheritance allows you to conceive of a new class of

objects as a refinement of another, to abstract out the similarities between classes, and to design and specify only the differences for the new class. Inheritance also supports the concept of code reuse, since each subclass that uses inheritance gains the behavior and attributes of the class above it. The new class may not even use all of the behavior it has inherited, which is just fine. The main point is that no redundant coding has to be done.

1. Abstract Classes

Not every class creates instances of itself. Inheritance can be useful for factoring out common useful behavior. Using our example of vehicles, suppose that the vehicle class, while containing the attributes and behaviors common to all vehicles, does not exactly describe any specific vehicle you might ever see. Its subclasses would be two-wheel drive, four-wheel drive, recreational vehicle, and maybe even motorcycles. The hierarchy would appear as in Figure 1 below:

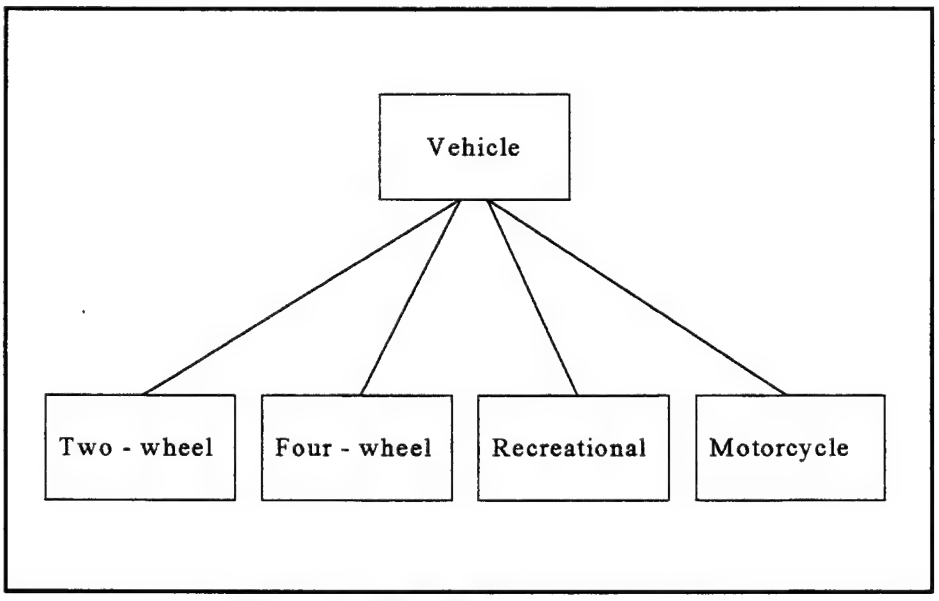

Figure 1. Class hierarchy example

Classes that are not intended to produce instances of themselves are called abstract classes. They exist merely so that behavior common to a variety of classes can be factored out into one common location, where it can be defined once and then be reused as needed. Abstract classes fully specify their behavior, but they need not be completely implemented. Concrete subclasses inherit the behavior of their abstract superclasses, and add other abilities unique to them. Occasionally, a subclass may even need to redefine one of the default implementations of its abstract superclass in order to behave in a meaningful way. Very often, subclasses are referred to as 'IS-A' relationships, as in "motorcycle IS-A vehicle" and "four-wheel drive IS-A vehicle".

D. OTHER RELATIONSHIPS

Examining the relationships between classes can be useful in determining the collaborations, or interactions between a client object and a server object. Three primary relationships in addition to 'IS-A' are occasionally useful:

- the 'HAS-A' relationship
- the 'Has-Knowledge-of' relationship
- the 'Depends-upon' relationship

Whole-part relationships are often indicated by such specifications as "Cars are composed of Parts". Classes such as Cars that are composed of other classes are referred to as composite classes. A composite class is responsible for containing the objects that compose it because it has a larger responsibility, such as managing its parts in some way or

maintaining some relationships among them. A composite class is, of course, responsible for knowing exactly what elements or objects it contains. Just as a real car relies on several of its parts to perform its function, a composite, or container, class relies on its components to interact in a specified way to perform its purpose.

The relationships 'Has-knowledge-of' and 'Depends-upon' are much more subtle. They typically imply interactions between objects that would seem on the surface to violate the principle of information-hiding. For instance, although a steering wheel and the wheels themselves would be modeled as separate objects, a steering wheel nonetheless needs to know where the wheels are and how to control them.

E. OBJECTS AND NETWORKS

Creation and application of objects in a standard framework can be simple enough on a single computer, but when an Object-Oriented system is expected to perform across a local or even wide area network, several difficulties arise. First of all, the operating system itself must be able to handle requests from remote hosts for access to an object framework and must also know where to go to get objects that its own applications might want.

1. CORBA

The Common Object Request Broker Architecture (CORBA), designed by the Object Management Group (OMG), is a specification for the handling of object requests across networks of any size. While it is possible to write and even use CORBA compliant applications, there is currently only one major operating system with CORBA built in: Orbix. CORBA is critical to the development of Command and Control planning systems, since

various commands in widely separated geographic areas need to be able to access common plans and be assured that the information they see is the same as every other commander. CORBA will allow any authorized agent to modify and update plans and know that the changes will be disseminated instantly around the world.

2. Persistence

One question that inevitably occurs when discussing Object-Oriented systems is "Where do the objects go when they are not active or being accessed?" Obviously there needs to be some mechanism to ensure that objects are not just lost in the ether of electrons. The most common solution is to employ an Object-Oriented Database (OODB) in order to store various objects until they are needed for query or updating. The creation of OODBs is itself a complicated topic. This notion of persistence, or permanence of an object, is sometimes also the responsibility of the object's operating system. For instance, ORBIX has its own methods of ensuring persistence for the objects that it manages.

F. MODELING THEORY

There have been numerous books written on the subject of modeling theory and how to best create object models out of existing physical or notional environments. Some expound a pure physical object method, while others extoll the virtues of viewing information systems as a completely different type of environment. All these theories have certain central concepts in common. The primary method for identifying objects in a system is through abstraction. All people view parts of the world through mental models, which help them understand and better interact with the environment. Just as a map must be significantly

smaller than its territory, and include only carefully selected information, so mental models abstract out those features of a system required for understanding, while ignoring irrelevant features. This process of abstraction is natural to humans, and affects how one understands the world. (Wirfs-Brock et al., 1990, pg. 3)

Abstraction itself can be used directly in the modeling of systems. As a first step, examine the specifications or attributes of a system and write out a list of every physical object and required action that comes to mind. Since this is a first draft, everything should be included. Next, the list must be examined to determine which items are redundant, or can be captured with a single concept. This brings the list closer to the final classes which will be used. Next model user interfaces to the system. This will entail use of most of the actual physical objects in the system. To complete the modeling, identification of abstract classes is extremely useful. Abstract classes are those which embody common aspects of one or more other objects in the model, and serves to provide a common class from which instantiations and specializations can be made. The abstract class itself is rarely if ever instantiated. While any system can be modeled using a straight object identification process such as this, it is often observed that, due to the lack of some sort of information system framework schema, the objects in the model must attempt to encapsulate all of the system's behavior. The development of the Core Plan Representation in Chapter III provides a framework for development of related systems. By its nature, it provides the assumptions and structure common to all plans, and thus allows abstraction of much simpler objects.

III. PLANNING BACKGROUND

In this chapter, we will review some of the current conventions used in planning, including OPLANs and JOPES, and cover some current C4I systems which include those functions as well as added capability.

A. OPLANS

OPLANs are the basic comprehensive unit in our current planning system. While their basic format is designated in Joint Publication 2, the great variation in possible situations could not be covered in any one document. OPLANs are therefore very free-form documents in order to ensure adaptability. While this helps in the development and writing of the plan, it can cause serious problems later as the plan is promulgated and expanded. What is needed is a mechanism to allow the current flexibility in plan creation, while simultaneously keeping a tracked format which can be understood and automated by software systems.

1. OPORDs

Operational Orders (OPORDS) are created in furtherance of the original OPLAN, although some may also be written for unrelated operations situations. They contain significantly more detailed information than the OPLAN and are disseminated in order to establish the specific procedures for certain operations. OPORDs are intended to be the official documentation used in order to carry out these operations, and they are used as such by the Army, Air Force, and Marine Corps. The Navy, however, has come to rely more heavily on what are known as messages of Commander's Intent. These messages are sent out

to all commands and are understood by the Navy to be the actual source of procedural information. It happens occasionally that these messages of Commander's Intent are not in strict accordance with the official positions, procedures, and intents that are written in the corresponding OPORD. This has caused considerable consternation when working with the other services who rely on the official OPORDs in order to ascertain the Navy's intentions and procedures. With an Object Schema in place for the development of OPORDs from OPLANs, the Navy will be far less inclined to rely on pseudo-formal methods such as the messages of Commander's Intent. Instead, the entire common shared system for plan creation will be available to them, as well as any other service that needs to coordinate with the Navy.

2. TPFDDs

The Time Phased Force Deployment Data (TPFDD) is also created along with the associated OPLAN, and contains all data and schedules regarding the transportation of equipment, supplies, personnel, and command structure to the appropriate theater of operations. In the past, the TPFDD printouts were potentially very lengthy, and did not lend themselves to easy extraction or analysis of the data contained therein. The development of JOPES on the Global Command and Control System (GCCS) has seen a drastic improvement in the way in which TPFDDs are handled. All the data is submitted in electronic format to the system, which can then analyze the data in any way necessary as well as create graphical time charts to present the data in a more easily understandable format. The benefits to be gained from an Object-Oriented treatment of TPFDDs lie along the lines of improved modularity of the data contained in a TPFDD as well as better distribution of any part of the

schedule by means of the CORBA architecture that controls the access of objects on the network.

B. JOPES

The Joint Operations, Planning, and Execution System (JOPES) has been the primary method employed by planning and operations communities to distribute OPLANs, TPFDD data, and interact with each other in furtherance of their objectives. The GCCS implementation of JOPES has brought several advantages over the older WWMCCS-based versions. Planners can now interact in a more real-time environment. They have the ability to not only 'chat' with each other, but they can also exchange figures or other graphical data instantly. Although GCCS has given a more graphical feel to JOPES, it has not significantly changed the way planning is envisioned or done. While the majority of Object Planning Schema is intended to remain transparent to the user, the capabilities that are inherently added by an object-based system will encourage planners to begin performing basic plan generation in a more modular and cooperative manner.

C. CURRENT C4I AUTOMATION

The following systems represent the most widespread implementations of systems that are intended to provide large-scale functionality. That is, both systems were created to provide tools and environments for the performance of such tasks as planning, unit tracking, and status analysis.

1. JMCIS

JMCIS is the Joint Maritime Command Information System. It is a direct result of the

combination of the Navy's Operations Support System (OSS) and several Joint functions such as the Joint Status of Readiness and Training (JSORTS) system. Most familiar to users of JMCIS is the Chart subfunction. This successor to the Joint Operations Tactical System (JOTS) consists of a fully configurable map of the world. On this map can be displayed a great number of things according to the filters set by an individual user. Among the possible things that can by displayed are icons representing individual ships, ground and air units, tracks representing the movement of units, and freeform quadrants. The way in which all these things are displayed is also highly configurable. Unit names can be decluttered (un-overlapped in crowded areas) in order to increase readability of the screen, and any unit icon on the display can be double-clicked in order to bring up a dialog box containing comprehensive and specific data about that unit. The data from which the Chart unit positions are taken is updated at regular user-defined intervals, and with the use of the Secret Internet Protocol Router Network (SIPRNET), units involved in actual maneuvers can be observed in near-real-time. This of course assumes a connection with the command flag ship in the theater of interest. JMCIS has several subfunctions related to the normal tracking of unit operations and conditions. These include facilities for monitoring Movement Reports (MOVREPS), Casualty Reports (CASREPS), Employment Schedules (EMPSKDS), and SORTS. While JMCIS, like GCCS, is a Graphical User Interface that uses Objects to interact with the user, its underlying code is not generally Object-Oriented. This, in addition to the fact that various part of JMCIS and GCCS were developed and programmed by at least five different contractors, explains the many headaches and incompatibility problems that are even

to this day surfacing.

2. GCCS

In 1993, the military began examining the various development efforts underway in the four services to decide which system to base the new Global Command and Control System (GCCS) on. The Navy's OSS was chosen, and GCCS still resembles it in many ways, except that GCCS is much wider in scope in order to encompass all elements of Joint planning and situation tracking. GCCS has done well in keeping with current technology, especially at time when software can become obsolete very quickly. As an example, GCCS uses a Web /HTML based network on the SIPRNET to convey the bulk of all information and connect all users of the net. GCCS can be seen as the first system to seriously address the problem of stove-pipe systems that has always plagued the military and adversely affected interservice communication and cooperation. GCCS contains JMCIS as its Common Operating Environment (COE), so all extra GCCS functionality was built around the existing JMCIS. Unfortunately it suffers from the same drawbacks. Since Object-Oriented systems provide a common pool of reusable objects from which many systems and subsystems can be developed, they can significantly reduce any problems with incompatibility caused by multiple source code programmers.

IV. DEVELOPMENT OF THE C2 OBJECT SCHEMA

The Object Model Working Group (OMWG) is based at the Armstrong Laboratory at Wright-Patterson Air Force Base in Ohio. It was convened in order to carry out coordination of the Joint Task Force Advanced Technologies Demonstration (JTF-ATD) Command and Control Object Schema Taxonomy. The OMWG operates under the sponsorship of the Defense Advanced Research Projects Agency (DARPA). In this chapter, the author will discuss the expectations for the C2 Object Schema project, give the framework for this effort, and explore the planning segment of the schema.

A. SCHEMA VISION

The long term vision for the C2 schema is for a single, widely distributed object schema shared among all DoD programs and systems. Following Object model principles, the schema would be implementation independent, with a large set of tools for searching and retrieving elements of the schema. The requested information could then be mapped to the requestor's system in a manner consistent with its architecture and context. The domain experts in each area of the schema would be responsible for the verification and validation of the objects created for their use, and will occasionally have reason to create their own classes. In addition to a base population of schema classes, it will support dynamic information and knowledge sharing through schema objects. In addition, the implementation-independent nature of the schema will allow programs to be constructed without specifics of the data in which the program is working, allowing the schema to grow without affecting current

programs (Object Model Working Group, 1996, pp. 15-16). The schema will be replicated at local sites, thus improving bandwidth efficiency since the only information transfers will involve standard object-to-object messages and the occasional object upgrade. The process of populating the schema with enough objects so that it reaches a critical mass capable of self-sustainment is an extremely large task and will take many years. Once this critical mass has been achieved, the number of reusable objects will be sufficient to allow meaningful use of the schema, and the development of specific applications will proceed at an accelerated rate.

B. THE C2 TARGET SCHEMA

The Target schema attempts to cover all aspects of command and control, and all the objects and concepts associated with it. Figure 2 shows the partitioning of the schema.

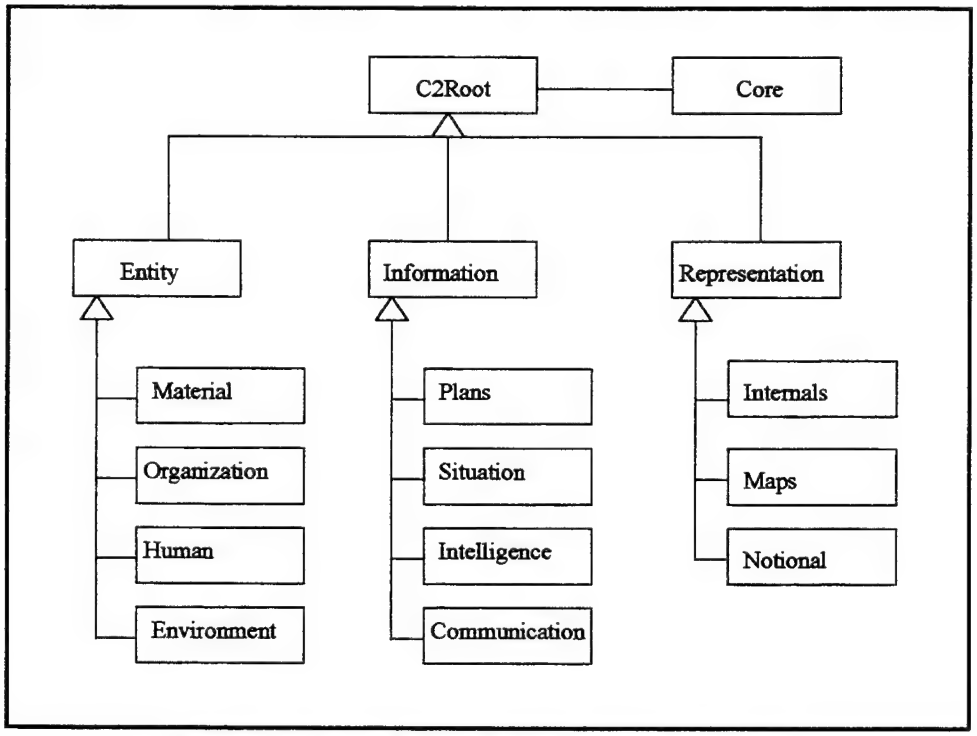

Figure 2. C2 Target Schema (OMWG, 1996)

The C2Root object is the parent to all other objects, although it serves mainly an administrative purpose. Plans are central to command and control, and as such serve as the primary collection points for information and intentions. Plans are complex aggregates which are meant to be dynamically changing as the plan evolves and knowledge is expanded. The plans component of the schema captures the plan itself, labeled c2Plan, as well as some support classes which do not reside in other parts of the schema. A plan contains numerous components, most of which are actually sequences or collections of objects. The other parts of the schema, such as situation and intelligence representation, are currently being developed and populated with objects by their respective domain experts and contractors.

C. THE CORE PLAN REPRESENTATION

The Core Plan Representation (CPR) is an attempt to build a multi-level core plan structure upon which many different plan representations can be developed. The main difference between the CPR and the core schema is that the CPR encapsulates meaning as well as information. The CPR is intended to support the representation needs of many different planning systems. It is also intended to use common functionality to facilitate the reuse and sharing of information between a variety of planning and control systems.

1. Background

The design of the CPR was derived from years of research in related fields. Planning is a fundamental component of intelligent behavior. The discipline of planning has been studied for generations in an attempt to produce more effective plans. Modern developments and techniques from the fields of artificial intelligence, operations research, management,

decision theory and philosophy have all been applied to planning problems. The areas of planning and scheduling are of particular interest in the artificial intelligence field and are defined as follow:

- Planning - Specifying a set of actions in order to meet a set of goals or objectives
- Scheduling - Resolving the dependencies among actions and resources in a plan to specify amounts of resources used over time and times at which actions will take place.

The CPR will allow progressive, hierarchical planning. This means that leadership and staff will use a planning application to develop guidance for their subordinate commands. This guidance includes background on the situation, objectives which must be met to contain the crisis, constraints on the actions of the forces and some specification as to the schedule of operations. This information is passed to individual commands which have specific requirements and methods of planning. The individual commands will then use the higher level base information plan to develop their own detailed plan which need not be of concern to the higher echelons. Finally, the CPR, when used correctly and with sufficiently detailed information, can be used in the creation of automatic simulations. For instance, a pilot would be able to use the plan as-is, in conjunction with the proper software, to run a simulated bombing run and observe all potential problems and opportunities associated with the strike. (Pease, 1996, pp. 3-4)

2. Building the CPR

The first step to building the Plan Representation is to identify those concepts

necessary to represent any plan. These concepts are:

- Action
- Actor
- Objective
- Resource

Actions are performed by Actors. The motivation behind performing the action is typically some Objective, although Actors may sometimes carry out routine subActions with no formal Objective attached. In performance of the Action, the Actor may utilize some resource such as fuel or parking space. The CPR at this point is shown in Figure 3.

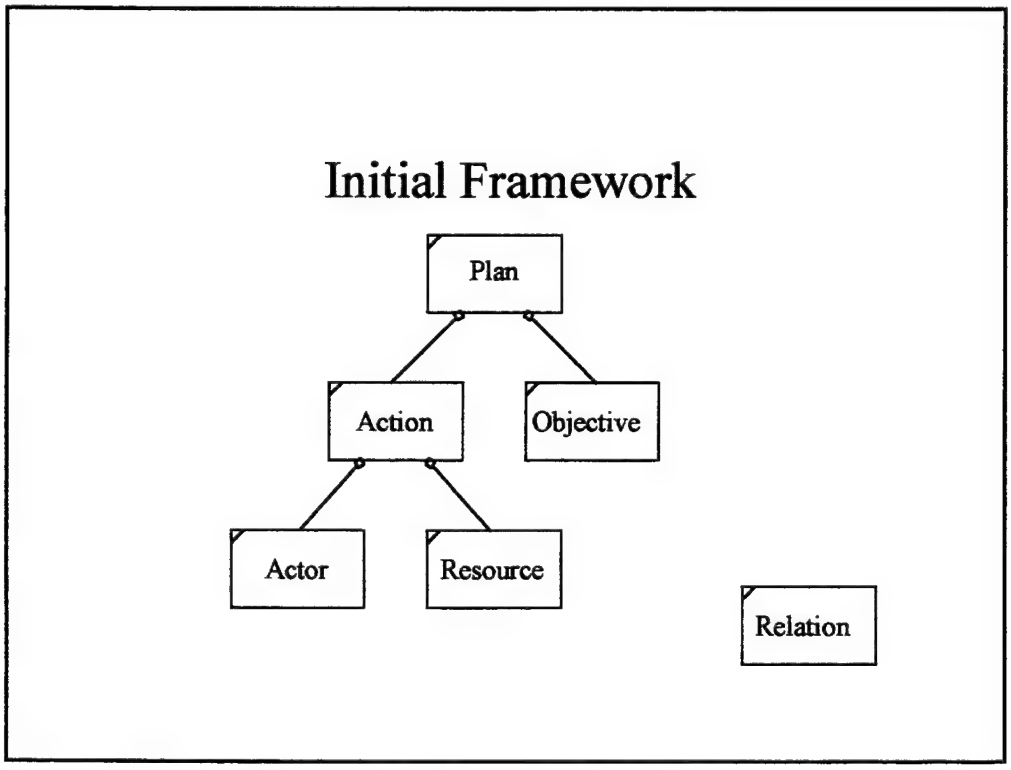

Figure 3. Rudimentary plan objects

Since the objects in Figure 3 are common to all plans and contain a base set of attributes, it makes sense to make them into classes, as described in Chapter II, for future instantiation and specialization. The classes shown are in a standard Object Model format. A diamond on the connector line indicates a composition relation, that is, a HAS-A relationship. Plans have Actions and Objectives, and Actions have Actors and Resources, for example. Next, concepts of workflow and manufacturing must be added to the representation to implement the notions of time, as well as spatial location. The classes TimePoint and SpatialPoint are added to satisfy this condition. TimePoint can be viewed in several ways. It can be a single point in time, but it can also represent the commencement of some activity or it can indicate a time interval in which some action must be accomplished. Similarly, SpatialPoint can be used to represent either a specific location such as a ship's home port, or a vague geographical area, depending on the need. Also needed is the class Relation, as shown in Figure 3, which will later be used to create Constraints between elements of the set, including self-association and sequence. Timepoint, SpatialPoint, and Relation have no specific place in the CPR, as they can be used as attributes in any of the existing classes of the model. These classes of the core are not typically meant to be directly instantiated, but rather serve to better understand the essence of a plan. When actually creating specific plans using the CPR, specializations of the aforementioned classes will be developed. In this way long and complex plans can be abstracted into specialized objects, and subclasses can be developed as needed to any level of specificity desired. At this point, the CPR includes all the classes

desired for the level of detail best suited to begin the abstraction process. The modified CPR is shown in figure 4.

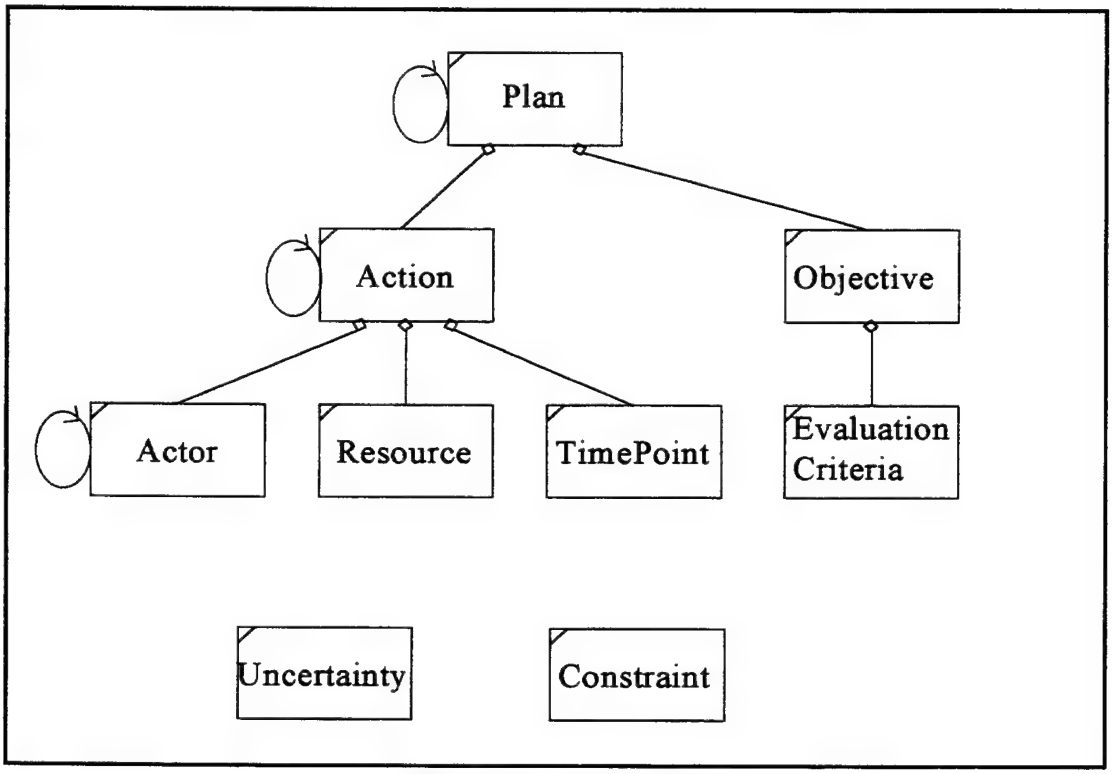

Figure 4. Basic CPR

Notice the self-association indicated on the Plan, Action, and Actor classes. Due to the aggregate nature of plans, it is appropriate to allow a Plan to be associated with another Plan, where one would represent the parent plan and the other a sub-plan. Similarly, the Action and Actor classes have a need for representing sub-actions and sub-actors, respectively. The purpose of the EvaluationCriteria class is to serve as a method to review the Objectives as measures of effectiveness of the Plan, or as a means to determine when the

Plan itself is actually complete. This is to be differentiated from the Evaluation attribute of the class Plan, shown in Figure 5, which is merely a textual description of the Plans' advantages over other proposed plans. Constraints and Uncertainty have also been added to the CPR. Constraint captures the conditions placed on Activities or Objects. It will also be used often as a means to indicated sequences of Actions. Uncertainty captures the degree of confidence in information. Constraint and Uncertainty can be included as elements of any object within the CPR, and so are shown with no connections.

To finish the CPR, a few remaining concepts must be added to satisfy completeness of its intended use. The class Plan needs some structure to contain metadata related to the plan. This metadata could contain information regarding the informal status of plan creation, or any number of other comments or notes relevant to the plan. For this purpose the class Annotation was created. PlanObjects are defined to represent entities referred to in an Action which are not the Actor or a Resource. For example, the recipient of a mail message would be recorded as an instance of PlanObject. Certain other classes included in the current version of the CPR are in rough form, and may be further refined or specialized in the future. Among these classes are Uncertainty and Imprecision. It is understood that both will require a type, measure, and source attribute, although there is currently no common representation scheme for the latter two. (Pease, 1996). Figure 5 on the following page shows the current complete CPR.

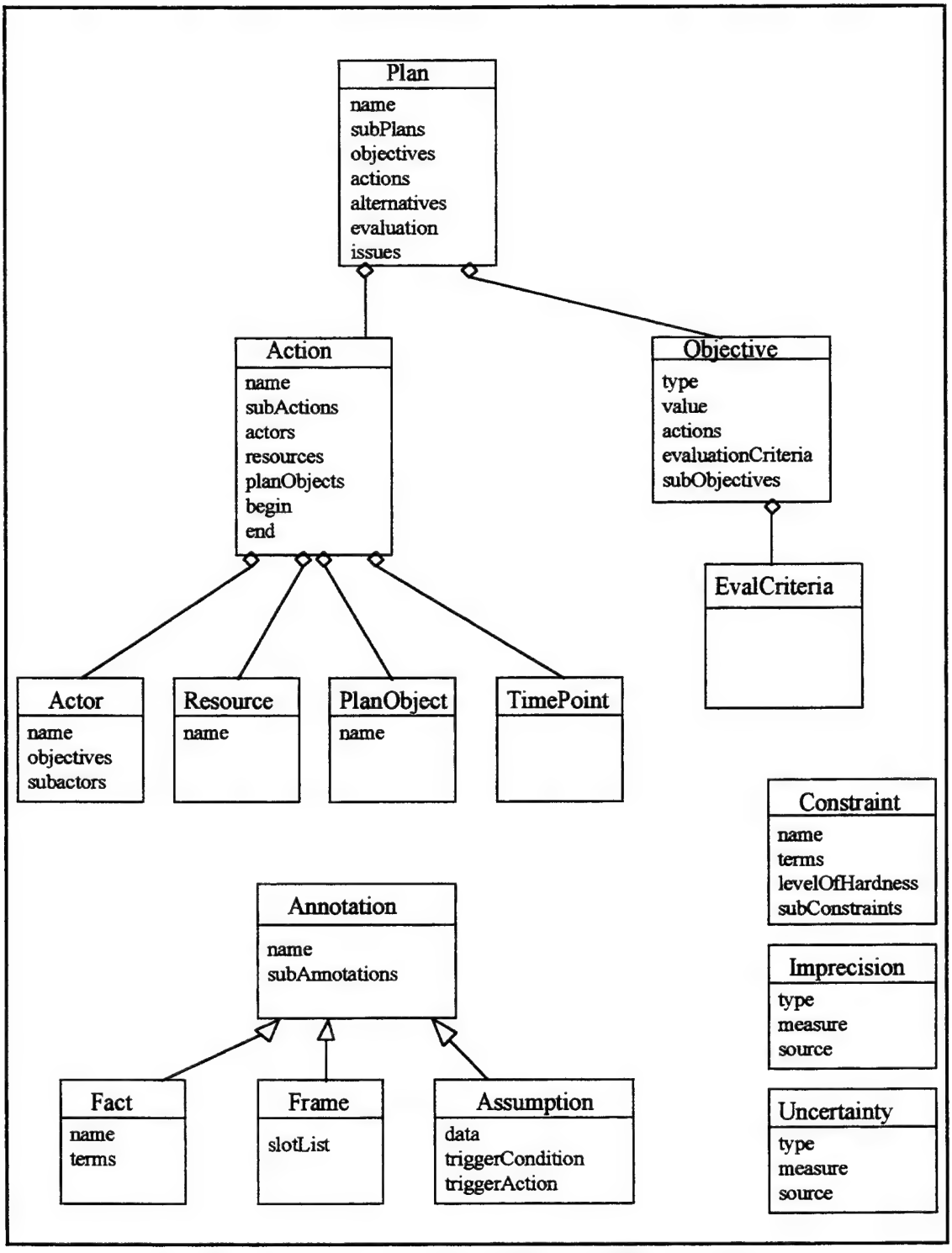

Figure 5. Completed CPR (Pease, 1996, pg. 13)

Now attributes of the individual classes can be examined. Action requires a name, as will all other primary classes within the CPR. Actors may have objectives of their own and so a reference to the plan objectives is added and called objectives. Since the Actor may not be a single person, but represent an aggregate of some kind, the attribute subActors was added. The Objective class already contained type, subObjectives, and evaluationCriteria, and value and actions are added to reference the list of Actions in the Plan itself. Constraints may include subConstraints, and also have an associated levelOfHardness which is an indication of that constraint's relative priority to other constraints in the Plan. Assumptions contain their associated data, as well as a condition for action, and that Action which is instantiated upon the related trigger.

Although the current CPR represents a number of trade-offs in order to achieve the desired level of simplicity, new classes and specializations thereof can always be created to fill any needed functionality in a plan derived from the CPR. For instance, in Chapter IV, a new class modifying Action will be created in order to simplify the specification of ordered lists applied to routine activities.

V. AN OPLAN MODEL

COMSUPAC OPLAN 5050 REV A (COMSUBPAC, 1990) is an unclassified OPLAN that promulgates the procedures to be followed for Submarine Search and Rescue operations. OPLAN 5050 first defines the SAR command relationships by describing the functions of the SAR Coordinator (SC), the SAR Mission Coordinator (SMC), and the On Scene Coordinator (OSC). These staff functions will become the primary Actors used in modeling Submarine Search and Rescue, and mapping it to the Core Plan Representation. After defining some assumptions and the mission statement for the OPLAN, the conditions for commencement of procedures are given. This plan consists primarily of three main Actions, the first of which is initiated upon the realization that a submarine has not filed a required accountability report. This first action is called SUBLOOK. This SUBLOOK procedure is a check on possible communications failure and is initiated by the SUBOPAUTH, who is usually also designated the SAR Mission coordinator. Action SUBMISS follows SUBLOOK if necessary. It is enacted when certain conditions are met in addition to the completion of SUBLOOK. Action SUBSUNK similarly follows SUBMISS or can be immediately commenced upon direct evidence of a sunk sub, such as the sighting of submarine wreckage or survivors, a red flare sighting where submarines are known to be operating, or reception of a distress signal by sonar or emergency radio buoy.

A. CONSTRAINTS

The model in this chapter for Submarine SAR is presented at two different conceptual levels. This is done primarily to avoid cluttered graphs, and also because the two concepts are sufficiently unrelated. The model abstraction shown in Figure 6 shows the events SUBLOOK, SUBMISS, and SUBSUNK in relation to the Plan 'OPLAN 5050'. More importantly, it demonstrates the usage of Constraints and subConstraints as methods of indicating commencement of an Action. SUBLOOK for instance, is initiated when the condition MissMsg is TRUE.

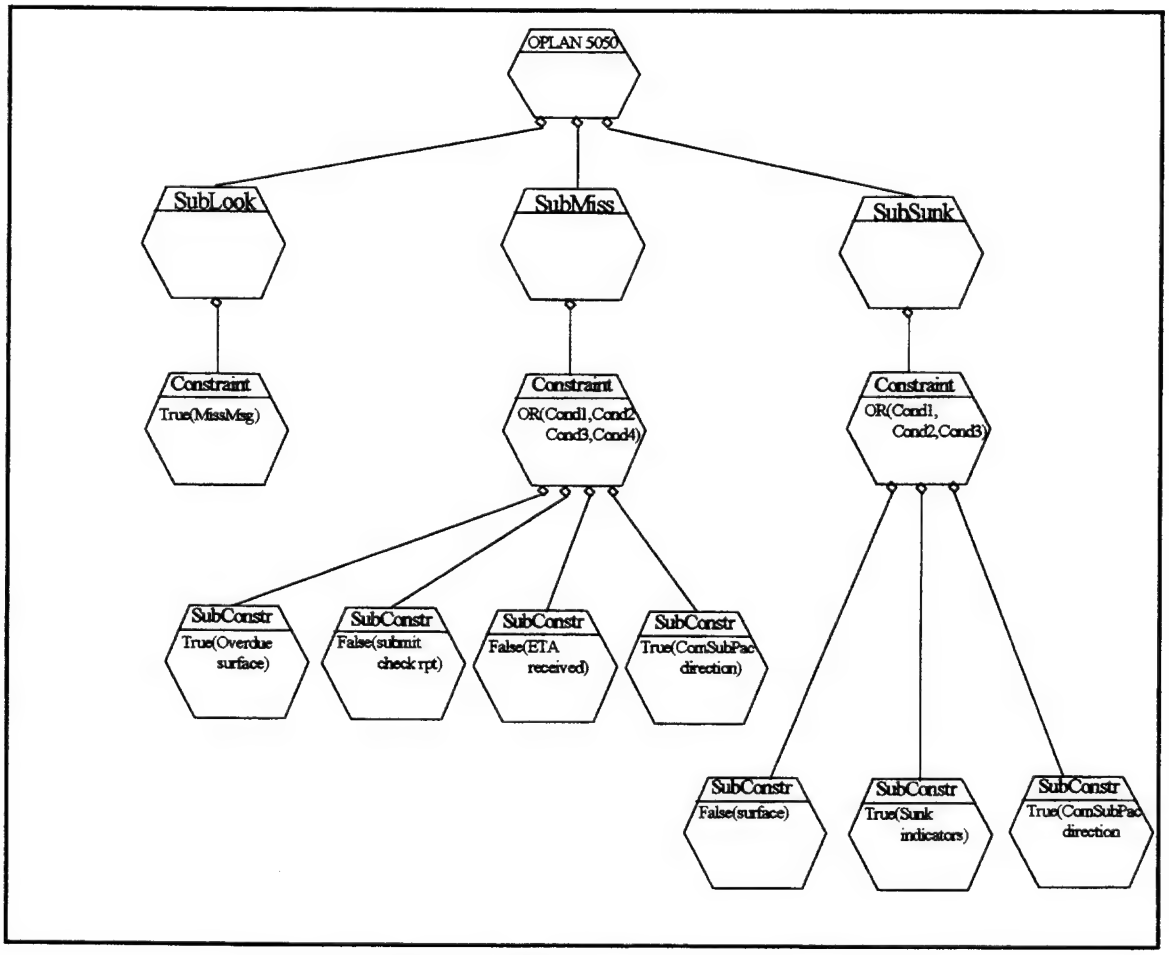

Figure 6. OPLAN 5050 constraint representation

The Constraints for SUBMISS and SUBSUNK are slightly more complicated in that they require subConstraints. The conditions for initiating the SUBMISS procedure are interpreted from Figure 6 as follows: SUBMISS will be commenced when Condition1 OR Condition2 OR Condition3 OR Condition4 is true. Conditions 1-4 are listed in the subConstraints. These conditions are listed; however, their implementation is a matter left for programmers who will develop the methods or algorithms for accomplishing the behaviors specified in the Object Models.

B. SUBACTIONS

The second view of Submarine SAR deals with the actual procedures that are performed when any of the three primary actions are commenced. Each of these Actions; SUBLOOK, SUBMISS, and SUBSUNK, entail the completion of comprehensive checklists. In the original specification of the CPR, the Action class contains the attribute subActions. This attribute was written to be an unordered collection of possible subActions, and any notion of sequence for the subActions would have to be implemented by the use of time-based constraints contained within the subActions themselves. For instance, the second item in a sequence of Actions would have a constraint attribute written like this:

After(Action2.begin, Action1.end)

This prefix conventional notation would be interpreted to mean "Action2 begins only after Action1 ends". Every subAction in the model would have to contain a similar statement in order to assure its proper location in the sequence of subActions. Naturally, this could

become quite cumbersome with a large list of subActions, and any mistake in the placement or wording of the constraints could cause serious unintended consequences.

1. SequentialAction

In order to create a model more suitable for large lists of relatively simple subActions, the new class SequentialAction in Figure 7 was created. It is a specialization of the Action class, and the only difference is that the subActions Attribute is now of type OrderedList. This could be any sort of data structure, such as a linked list, in which the order of elements is maintained.

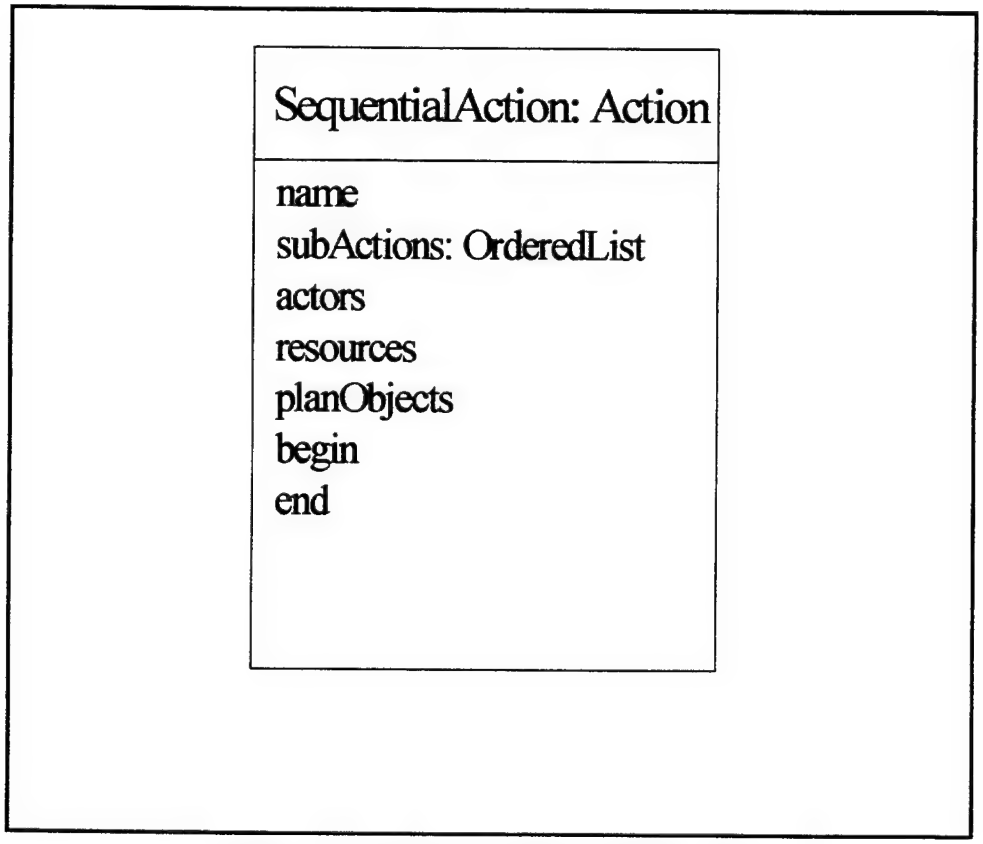

Figure 7. New class SequentialAction

As previously mentioned, using this new class can significantly improve the layout and comprehensibility of the graphical model. For sake of completeness, even though the sequential subActions of the primary Action are enumerated within the OrderedList, it is still necessary to show graphically how the subActions are related to Action. Since the actual constitution of the subActions can be delineated elsewhere, either graphically or textually, Figure 8 shows only the relations between the Action and subActions, and also begins to give some indication of how cluttered a model with many subActions can become.

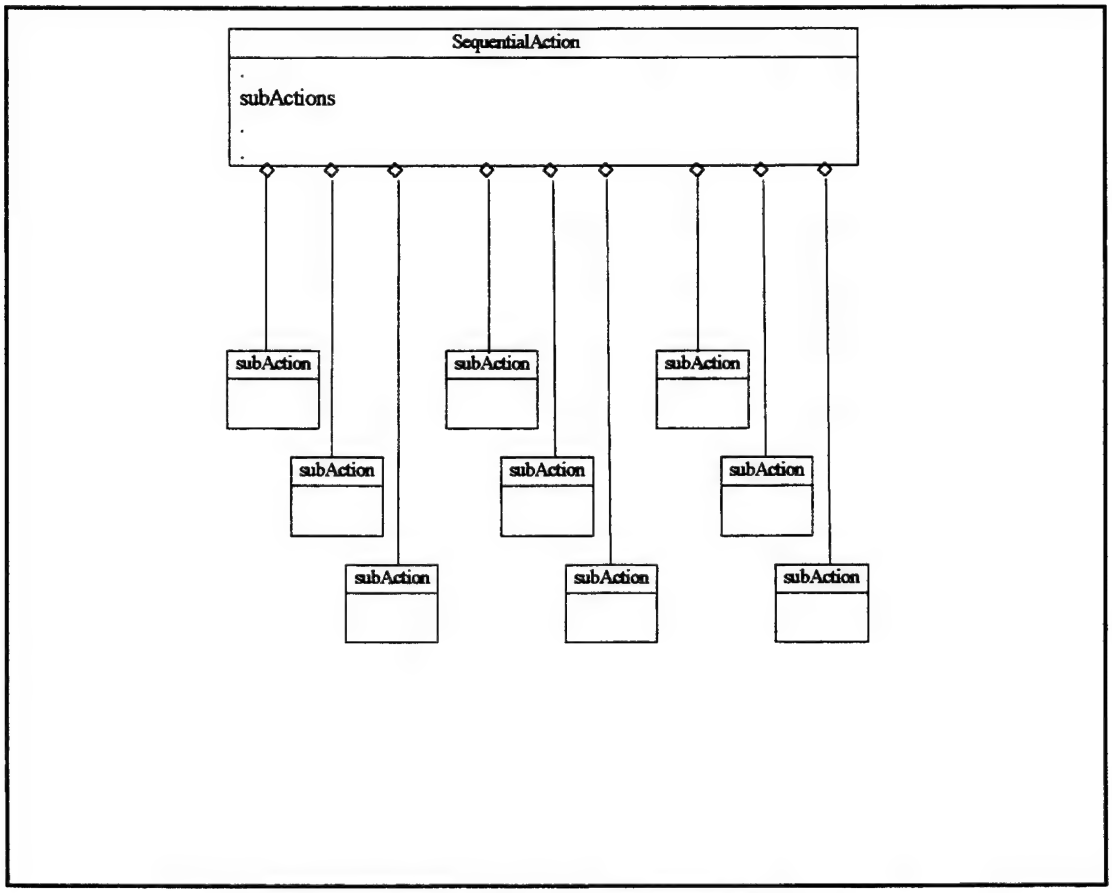

Figure 8. Example of multiple subActions

2. SUBLOOK

The event SUBLOOK is initiated when a submarine has not filed a required accountability report. This Action is considered to be generally less serious in nature than SUBMISS or SUBSUNK. When commenced, all seniors in the chain of command are informed of the circumstances by telephone, but the submarine is not considered missing. No surface ships or submarines will be diverted, nor any unit recall initiated. No press release is authorized. About 20 hours after commencing SUBLOOK, or when the situation dictates, events SUBMISS and SUBSUNK may be executed with the concurrence of COMSUBPAC(COMSUBPAC OPLAN 5050, Pg. 2). SUBLOOK consists primarily of two checklists, one to be completed by the COMSUBPAC Command Center and one for the SUBOPAUTH. Both other events also have two checklists to be completed by the same entities. The tasks listed and numbered on the checklists will become the subActions for the model. The subActions generally fall into three categories:

1. The actor performs some type of event logging
2. The actor prepares and sends a message
3. The actor notifies various other entities

The subActions for both checklists are virtually identical, therefore object reuse is especially helpful and it is necessary only to model one checklist. The complete class with attributes of Action SUBLOOK is presented in Figure 9.

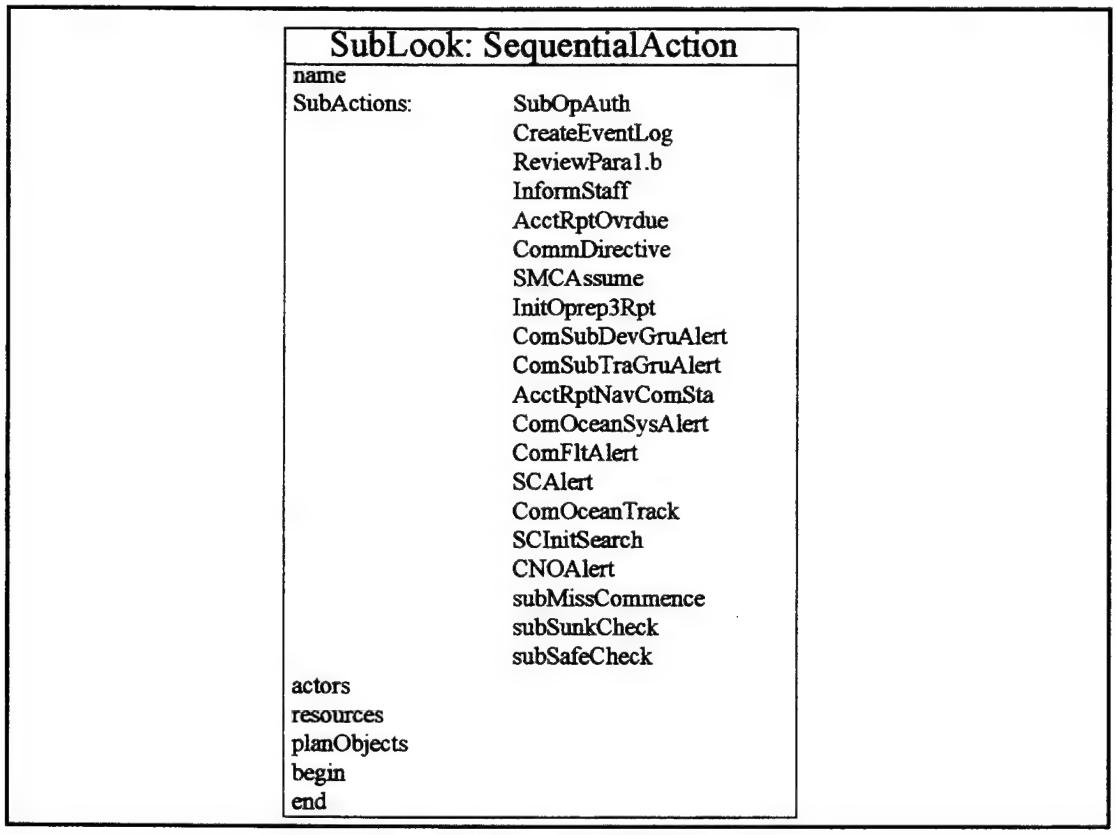

Figure 9. Action SUBLOOK

Normally, the attribute SubActions would be merely a link to the OrderedList in question, but the subActions are listed in the object of Figure 9 for clarity. There are 20 subActions identified in SUBLOOK, and considering the cluttered look of Figure 8 which has only 9 subActions, it is obvious how the SequentialAction class saves space. In addition, the new class avoids the laborious task of specifying the individual time constraints the subActions would otherwise have to contain. In summation, the Action SUBLOOK has three primary responsibilities: start a log of events, attempt to establish contact with the submarine, and begin notification of several commands. Since all subActions are merely instantiations

of the class Action without any specialization, it is only needed to model one subAction for reference and to demonstrate how Actors, Resources, and DomainObjects (formerly planObjects) fit into the model. Figure 10 shows an instantiation of the subAction AcctRptOvrdue.

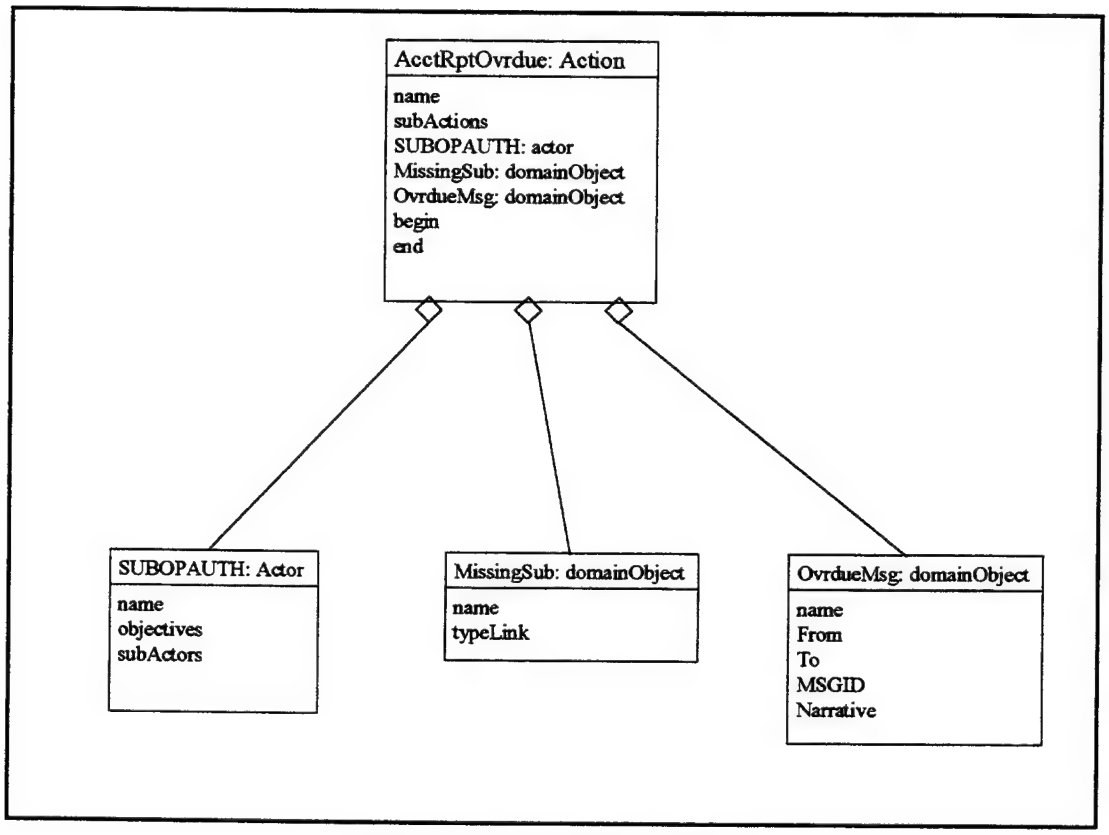

Figure 10. SubAction example

It is common to try to cast direct objects of the Actor as Resources instead of domainObjects. The distinction is that domainObjects can be anything they need to be in order to complete the notions required by the Action. A Resource is usually something that is consumable, such as fuel, or something that can be allotted.

3. SUBMISS

Action SUBMISS is also an instantiation of SequentialAction. This event is initiated when the constraints as shown in Figure 6 are met, that is, when SUBLOOK has been completed and the submarine is overdue in surfacing or reporting, when a particular situation warrants, or when directed by COMSUBPAC. Figure 11 below shows SequentialAction SUBMISS, and the subActions attribute shows the OrderedList of actions to be completed.

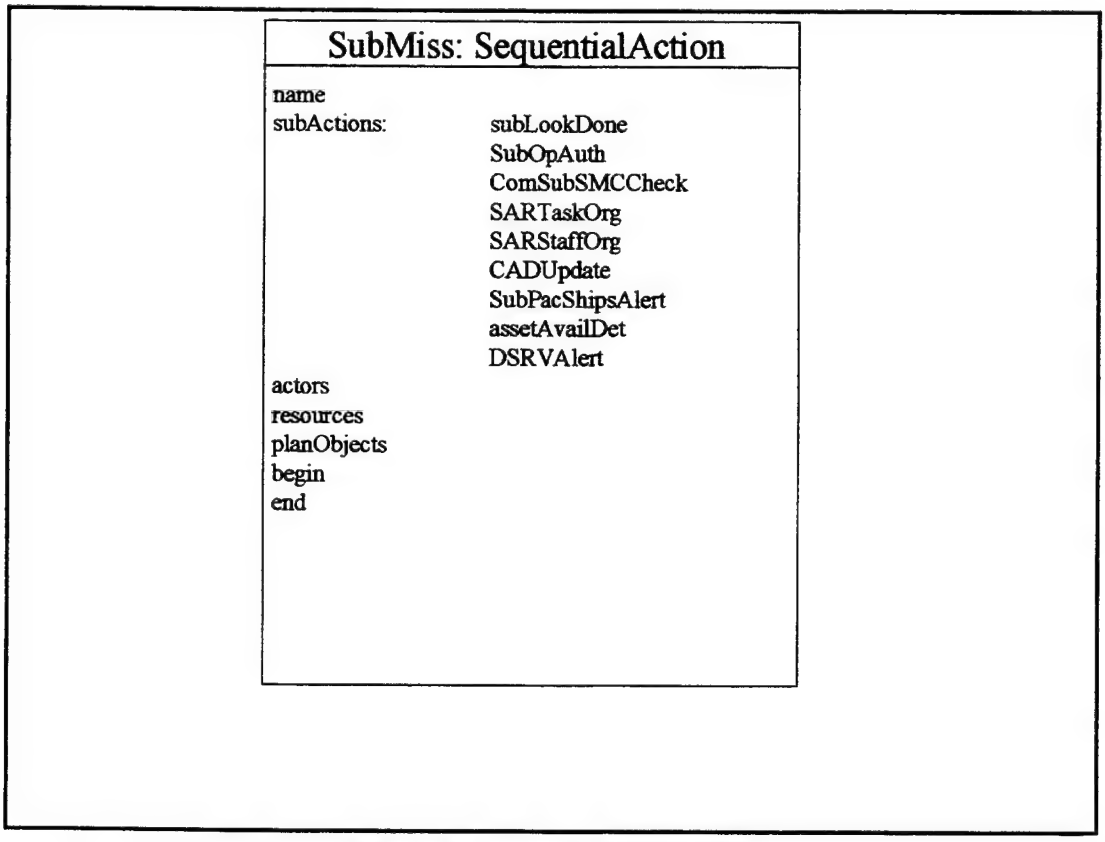

Figure 11. Action SUBMISS

As before, the sequential subactions are conveniently listed in order. Each subAction is merely an abbreviation of a checklist item given in OPLAN 5050. The subActions listed

are similar in function to those in SUBLOOK except for the establishment of the SAR task organization.

4. SUBSUNK

The SUBSUNK Action is more concerned with the movement of assets into the suspected area of recovery. Its final subAction is the commencement of standard rescue operations in accordance with Naval Warfare Publication 19-1. All of SUBSUNK's subActions are identical in representation to Figure 10, except that the domainObjects will be references to special ships and Deep Submergence Rescue Vehicles (DSRVs). Action SUBSUNK is show below in Figure 12.

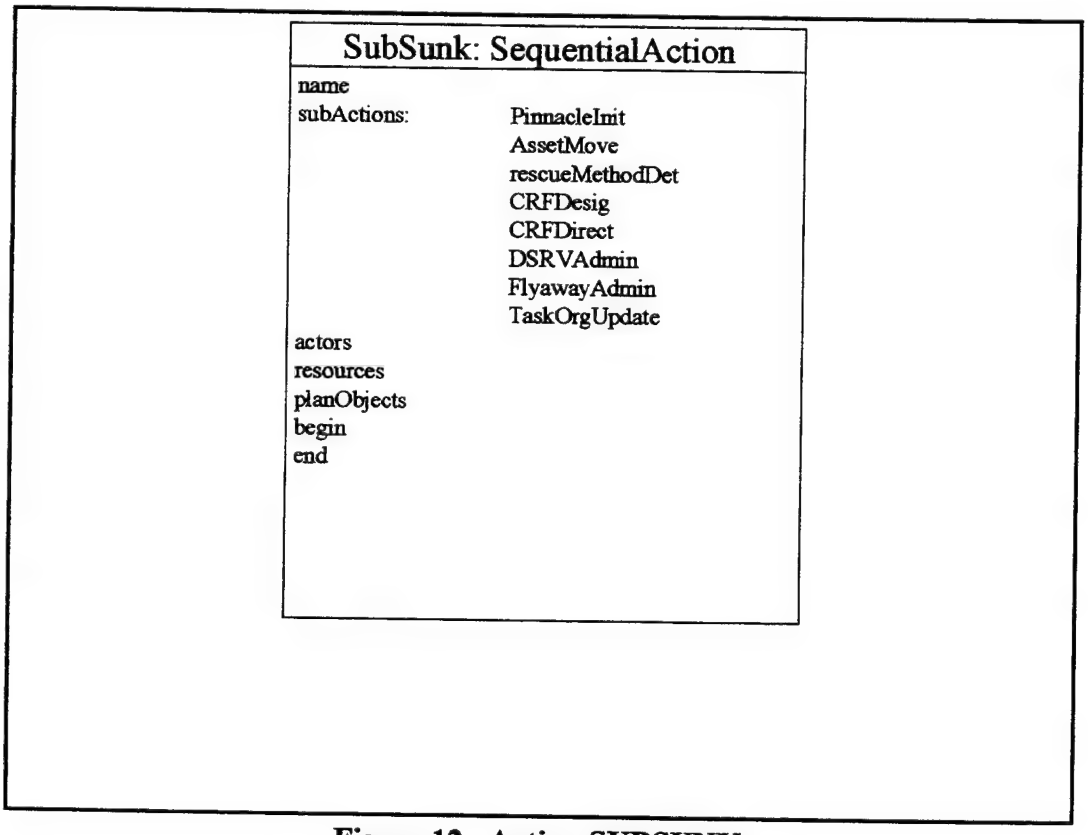

Figure 12. Action SUBSUNK

C. SUMMARY

Creation of the Object Model for OPLAN 5050 was relatively straightforward in nature due to its detailed lists and sequences. The value of 'thinking in objects' cannot be understated. At first reading, the OPLAN did not suggest clear objects or relations, but after studying the Core Plan Representation and understanding its concepts, it was a simple process to determine the pertinent Actors, Actions, domainObjects, and Objectives. It is also noted that each subAction of SUBLOOK, etc. is performed by the Actor with some Objective. However, the Objective is not as rich as that of the entire Plan itself. These small subObjectives can be viewed as merely the motivation to complete the appropriate checklist. Since the very notion of the SequentialAction necessitates the performance of the subActions, the subObjectives related to them are irrelevant or unnecessary.

VI. CONCLUSIONS

The anticipated timeline for sufficient population of the OMWG's Object Schema is between two and ten years (OMWG, 1996, pg. 21). This uncertainty is based largely on the unpredictable nature of expected support. The OMWG is currently exploring all avenues of obtaining help in completion of the Schema, including thesis work such as this. Past experience with Object-Oriented Systems built from the ground up have validated the benefits expected by the discipline. With the solid base in planning theory garnered from the fields of Artificial Intelligence, Operational Analysis, and Workflow Analysis, the C2 Object Schema has thus far done everything possible to ensure the solidity and flexibility of its products. The Core Plan Representation in particular was carefully designed to achieve just the right level of complexity. For example, the three sentences "There is a mammal on the porch", "There is a dog on the porch", and "There is a Toy Poodle in the middle of the porch" all convey the same meaning, but each is significantly different in the level of detail it provides. Consideration of this notion of level of detail was foremost in the development of the CPR, and been proven by the large number of instantiations that have been developed with it.

A. CONVERSION AND DEPLOYMENT

The OMWG, located at Armstrong Laboratory at Wright-Patterson Air Force Base, has assembled a team of planners and modelers who are working directly with some of the foremost idea people in the field. Austin Tate of the Artificial Intelligence Applications Institute at the University of Edinburgh, and author of Towards a Plan Ontology is a well-

respected and well-known figure in the world of AI. The OMWG has based most of their Schema taxonomy on his work, and current developers of the Schema constantly seek his help. The official strawman paper for the creation of the C2 Schema was promulgated only in September of 1996, and the project has gained steady momentum in a short time. The key element to the success of the C2 Object Schema, as with most all projects, is support by the command hierarchy and their commitment to its development. With the recent emergence of GCCS as the primary project of the Joint community, support and funding of other projects is likely to level for a time. The nature of the C2 Object Schema project, however, is such that fielding time is already expected to be in the distant future. This means that it can proceed at almost any rate of development while still making progress toward the eventual goal of a critical mass of reusable objects that will make any system using them self-sustaining. With the rapid advance of Object-Oriented analysis and programming in the commercial world, including applications such as JAVA and CORBA which support objects, the need in the United States military for a solid Object Schema is rapidly approaching.

LIST OF REFERENCES

Carrico, T. M., Command and Control Schema, Defense Advanced Research Projects Agency, 1996.

Pease, R. A., Core Plan Representation, Defense Advanced Research Projects Agency, 1996.

Wirfs-Brock, Wilkerson, Wiener, Designing Object-Oriented Software, Prentice Hall PTR, 1990.

-, Copernicus ...Forward, Chief of Naval Operations Strategic Planning Office (N6C), 1994.

-, GCCS From Concept to Reality, GCCS Division (J6V), 1994.

-, Naval Doctrine Publication 6, Naval Doctrine Command, 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor Dan Boger (Code OR/BO) 1
   Naval Postgraduate School Department of Operations Research
   Root Hall
   Monterey, California 93943

4. Robert B. Reeves 1
   110 Terraza San Angelo
   La Habra, California 90631